Multiclass Random Forest on FPGA using Conifer

Kevin Druart
UBO / Lab-Sticc
Brest, France
druart@univ-brest.fr

David Espes
UBO / Lab-Sticc
Brest, France
espes@univ-brest.fr

Catherine Dezan
UBO / Lab-Sticc
Brest, France
dezan@univ-brest.fr

Alain Deturche
Thales DMS
Brest, France
alain.deturche@fr.thalesgroup.com

***Abstract*—Random Forests are used in machine learning to build multiple decision trees using different feature sets, enhancing accuracy and resistance to overfitting. Random Forests originate from the Bagging method and can be used in many areas. Field-Programmable gate Arrays (FPGAs) offer a flexible and energy efficient platform. It is essential for real-time applications to have low-latency parallel processing. Random Forests are difficult to implement due to high hardware resource consumption and increased memory usage. To help developers to deploy their random forest models on FPGA, the conifer framework is usually used. This framework represents each decision tree from the forest as a boosted decision tree. This representation is useful for large decision tree and for being executed on conventional processing unit. In this paper, Conifer framework is extended to offer multiclass classification to Random Forests incorporating small and medium size trees. Comparison between rolled and unrolled random forest implementation on FPGA are proposed.**

***Index Terms*—Random Forest, Machine Learning, FPGA**

I. Introduction

Random Forests were introduced by [1] at the beginning of the 21st century and are applied in machine learning for classification and regression tasks. They build several decision trees using samples from the training data, each tree using a different set of features to find the optimal divisions. The final prediction is achieved through majority voting of the trees for classification or by averaging the predictions for regression. Random forests are an extension of the Bagging (Bootstrap Aggregating) methodology introduced by Breiman in 1996. The random forest algorithm involves the random selection of variable subsets, enhancing the model's predictive performance and robustness.

Random Forests are more resistant to overfitting and have higher accuracy than single-decision trees. They are versatile for various purposes like identifying spam, image recognition, text recognition, predicting prices, and analyzing economic trends. Besides, they can recognize the most essential features in a data set, rendering them a powerful and adaptable resource.

FPGAs (Field-Programmable Gate Arrays) are attractive for embedding AI due to their flexibility and ability to perform low-latency parallel processing, which is crucial for real-time applications. They consume less energy than CPUs and GPUs and can be reprogrammed to adapt to new tasks. Several frameworks, such as HLS4ML (High Level Synthesis 4 Machine Learning) [2], Xilinx's Vitis AI [3], Intel's OpenVINO , FINN, Conifer [4], facilitate developing and deploying AI models on FPGAs.

Implementing Random Forests on FPGAs presents certain challenges, as unrolling a tree requires significant hardware resources, and managing multiple arrays can lead to increased memory consumption. The Conifer framework, one of the primary frameworks for deploying boosted decision trees, represents each tree in the forest as a boosted decision tree. For this purpose, an array is allocated for each class, which contains only a subset of the nodes and leaves from the original tree. Unlike decision tree representation with only one array, multiple array representation of a tree usually consumes more hardware resources due to redundant nodes that are present in multiple arrays. Boosted decision trees are a representation that was conceived for CPUs, which have big memory (RAM) and many cores. Each boosted decision tree can be executed on a dedicated processor unit core, decreasing the overall inference time. However, this representation is not the best for FPGA, which can execute almost all tree nodes in parallel. Indeed, a tree can be fully deployed on an FPGA.

Our solution proposes expanding the Conifer framework and allowing for rapid and efficient FPGA implementation of any Random Forest (RF) model for multiclass problems.

Our contribution is twofold:

- adapting the Conifer framework to multivariable Random Forest classification
- analyzing the resource usage of various random forest implementations.

We will begin by exploring how the scientific literature tackles our problem in section II. Section III presents the Conifer design flow and our improvements, followed by a detailed description of the experimental setup in Section IV. The results are outlined in Section V, and this work will be concluded in Section VI.

II. Related work

Since random forests were first introduced and studied in the early 2000s, they have become a well-known algorithm in machine learning, as detailed in [5].

Improvements of the initial RF algorithm have been proposed, in particular by [6] who proposes Bernoulli Random Forests allowing the simplified construction of trees with two independent Bernouilli distributions.

Decision forests seek to increase a single decision tree's capacity for generalization by combining the output of multiple trees. In their review, [7] emphasize the methods and strategies utilized in decision forests, particularly the three main components of decision forest use—growing, thinning, and combining.

The implementation of Machine Learning algorithms in software has been significantly facilitated by the availability of frameworks such as SKLearn [8] in Python. An hardware implementation method has been developped by [9] for decision trees.

The authors in [10] show that their hybrid CPU-FPGA platform delivers significant speed improvements and energy savings compared to traditional CPU-only implementations, making it well-suited for large-scale data processing tasks. While there are some limitations in adapting to different types of decision tree algorithms, this approach provides a scalable and efficient solution for embedded applications that require high-performance decision-making.

Multi-valued decision diagrams used in [11] can significantly increase node count, resulting in higher memory usage and potentially slower training times. Their implementation approach uses the Altera SDK and OpenCL code.

In [12], they improve the performance of implemented decision tree classifiers by designing four different architectures to optimize the processing speed and energy efficiency. The evaluation results show significant improvements in processing time and energy consumption compared to traditional software implementations. However, these hardware accelerators have limited flexibility and may limit their adaptation to different types of decision tree algorithms.

Conifer is derived from HLS4ML [2] and originally enabled the implementation of Boosted Decision trees on FPGAs using the Xilinx [3] suite. For this reason, the way you use Conifer is very similar to the way you use HLS4ML, with the same file structure, similar configurations, and the benefit of the questions asked by the HLS4ML community. This framework takes over the conversion and implementation architecture of HLS4ML, while adding a wide range of conversion options starting from ONNX [13], TMVA [14], XGBoost [15], TensorFlow Decision Forest [16] and Ydf [16]. Conifer also enables the implementation of Forest Processing Units (FPUs), offering great flexibility by performing inferences on different Boosted Decision Trees without having to rebuild the IP or bitstream. The FPU can be compared to the DPUs in VitisAI. Thanks to the use of the Xilinx tool chain, Conifer offers a wide variety of implementation choices, from Xilinx HLS (High Level Synthesis) code generation, to VHDL, C++, python (for conversion validation) code generation, to bitstream generation and simplified execution with PYNQ [17].

III. Methodology

This section will describe how multiclass classification with Random Forests on FPGA has been achieved by modifying the open-source Conifer framework and the tree unrolling mechanism.

The implementation flow proposed by Conifer consists of 5 parts as depicted in Figure 1. First, an SKlearn model is created and trained. Next, a conversion module converts the functional model into an HLS model. It will then be written into a Xilinx project by the Writer. Compilation then takes place, and the RF can be implemented and run on the FPGA board. Our modifications have been validated only on the SKLearn conversion block.

We begin by implementing software experimentation models based on the multivariable Random Forests proposed by SKLearn. It is the functional model in Figure 1. We instantiate and parameterize this model with Hyperparameters (HP), such as the number of estimators or the maximum depth of the trees in the forest. Other parameters are available in SKlearn.

To train the defined classifier, we need a dataset. Using a standard SKlearn flow, we import a dataset that we preprocess to obtain a train and a test dataset, with an 80% train/test ratio. We use the train set (X_train and y_train) to train the Random Forest classifier. Once fitted, our model must be evaluated using metrics. In our case, we have chosen accuracy to evaluate a classifier because it is simple and easy to understand. It allows us to quickly compare the performance of an FPGA implementation with a software implementation.

Once our forest is trained and evaluated, we can feed it into Conifer to start the transformation. The first block is the SKlearn conversion part, where the forest is converted using the custom convert_RF function. This function has been modified to correctly map the individual trees into a python Ensemble Dictionary.

In the conifer implementation, each decision tree is represented by 6 arrays of size 2^n, where n is the tree depth. For a complete binary tree of depth (n): The number of leaves (terminal nodes) is (2^n). The total number of internal nodes (including the root and intermediate nodes) is given by the sum ($2^0 + 2^1 + 2^2 + \cdots + 2^{n-1}$). We can calculate the total number of nodes: Total number of nodes = $\sum_{i=0}^{n-1} 2^i$ This geometric sum can be simplified to: $\sum_{i=0}^{n-1} 2^i = 2^n - 1$. So, the total number of internal nodes is ($2^n - 1$). For the total number of leaves, Number of leaves = 2^n

The six-array representation appears to be directly extracted from the storage mechanisms utilized by the SKLearn library for decision trees. Children_left and children_right contain the left and right child indices for each node, with -1 indicating a leaf. The feature array specifies the index of the feature used for testing each node, while threshold indicates the threshold value for that feature. The n_node_samples array keeps track of the number of samples that reach each node. The value and impurity tables contain the prediction values for leaf nodes and impurity measures (such as Gini index or entropy) respectively. Finally, node_depth stores the depth of each node in the tree. Together, these arrays enable the decision tree to be efficiently represented and traversed for prediction and analysis.

To propose multi-class representation, we have to face two challenges: 1) to use the same structure and arrays proposed by Conifer and 2) to avoid increasing the size of the arrays.

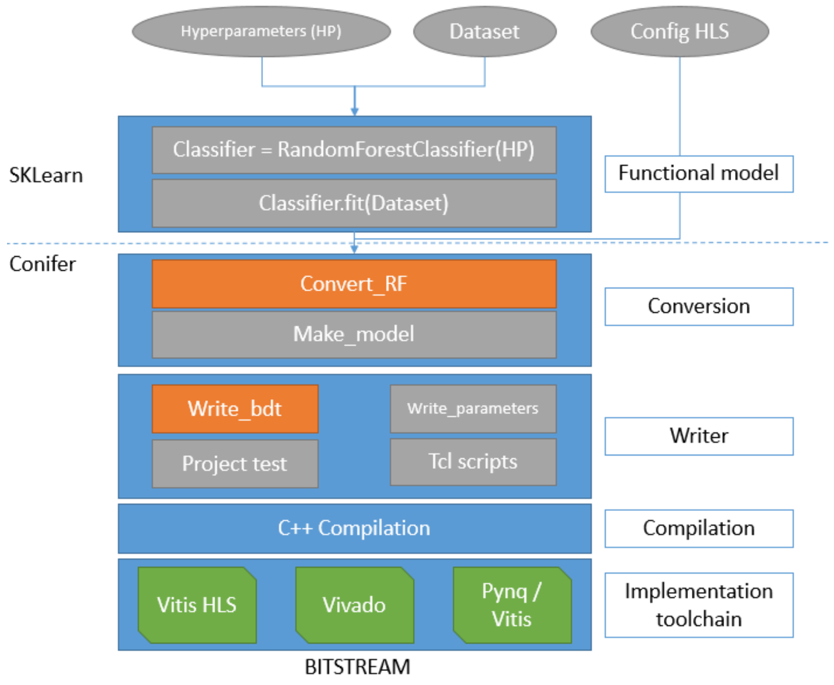

Fig. 1. Overall Conifer methodology. Orange blocks have been modified.

The first challenge is important to maintain the compatibility with previous Conifer versions. Conifer will use SKlearn to model the RF that will be deployed on FPGA. By adding new functionalities (i.e., multi-class inference), we have to keep the main arrays that are resulting from SKlearn. The trees, represented as six arrays, are imported directly into the HLS project using the convert_RF conversion method. SKLearn uses this structure for training, inference, visualization, optimization, etc. We need to focus only on the inference phase. It is, therefore, possible to optimize how trees are stored and accessed by taking advantage of the parallel architecture of FPGAs. The six arrays of size $2^n - 1$, as they stand, risk saturating resources frivolously. Vitis HLS optimizes this kind of problem to a certain degree. However, three arrays are used by SKlearn during the training phase. The implementation on FPGA will only be executed for inference. As such, we only keep three arrays (i.e., threshold, value and node_depth). Same if our approach is multi-class, the number of resources on FPGA is hence half.

For the second challenge, we change the representation of the value array. Initially, the value array was used by conifer to store the prediction values. To allow multi-class prediction, we modify the content of the value array to store the predicted class. Such a change is quite simple and do not require a new array to know the inferred class. Indeed, we adapt the vote function to compute the majority and decide which class is inferred from the RF.

The unroll optimization technique enables the simultaneous execution of all comparisons involving non-leaf nodes. By doing so, it gathers and stores the results of these comparisons in a dedicated comparison array. This approach enhances efficiency by processing multiple evaluations at once, rather than sequentially, ultimately improving the overall performance of the algorithm. The unrolling mechanism can be triggered during the Conifer configuration and will influence the implementation process of the decision functions.

IV. Experimental Setup

We chose the Xilinx ZCU102 evaluation board, based on the Zynq UltraScale+ MPSoC, which offers a powerful combination of FPGA and CPU/GPU resources. It features 600K logic cells, 2,520 DSP blocks, and 32.1 Mb of memory for programmable logic. The CPU features a quad-core Cortex-A53 for general-purpose processing, a dual-core Cortex-R5F for real-time tasks, and a Mali-400 MP2 GPU for graphics applications. The board also includes 4GB DDR4 SODIMM with ECC, 512MB DDR4 for programmable logic, and various interfaces such as PCIe Gen2 x4, USB3, DisplayPort, SATA, and 4x SFP+ for Ethernet. It is compatible with Vivado Design Suite and PetaLinux, making it ideal for automotive, industrial, video, and communications applications.

We used Xilinx Vitis 2023.2. On the software side, sklearn 1.5.1 was installed with Python 3.10.12 and our modified Conifer 1.4 version. Our machine is equipped with an Intel Xeon w5-2465x with 32 cores and 256Go of RAM on Ubuntu 22.04.

a) Dataset: We have chosen to utilize the CIC-IDS-2017 dataset to detect cyber attacks in IT environments. This dataset represents a standard network environment with various operating systems on computers and servers. It comprises more than 2 million samples, including 14 different types of attacks and benign network flow. 78 features describe each sample.

n_estimators	max_depth	acc_skl (%)	acc_hls (%)	lat_skl (μs)	lat_hls (μs)
2	2	0.88	0.87	526.91	0.72
2	4	0.94	0.88	637.05	0.8
2	5	0.96	0.95	524.04	0.88
10	3	0.94	0.93	597.24	1.6
10	4	0.96	0.96	602.48	1.9
10	5	0.97	0.96	747.44	3.7
10	6	0.98	0.98	628.23	5.67
20	2	0.88	0.88	793.93	2.44
20	4	0.96	0.96	786.54	3.36
20	5	0.98	0.98	774.38	6.9
50	3	0.94	0.94	1251.46	4.6
50	4	0.96	0.96	1247.17	7.56
50	5	0.98	0.98	1243.35	16.99
100	3	0.94	0.94	2017.97	8.6
100	4	0.96	0.96	1968.15	16.57
100	5	0.98	0.98	2001.05	33.49
200	3	0.94	0.94	3408.43	16.59
200	4	0.96	0.96	3428.22	32.57

TABLE I
COMPARISON OF SOFTWARE AND HARDWARE ACCURACY AND LATENCY (IN MS) DEPENDING ON RF HYPERPARAMETERS ON FLOAT PRECISION FOR ROLLED IMPLEMENTATION

b) Metrics: To evaluate our implemented Random Forest and the impact on performance caused by the implementation toolchain, we chose to compare the software accuracy given by SKLearn with the real accuracy obtained with the transformed model in C++.

The estimations supplied by the Xilinx toolchain were utilized to assess energy consumption measured in Watts. Additionally, resource estimations produced by Vivado, calculated after the routing process was completed, were taken into account. For hardware latency, the estimations generated by Vitis HLS provide valuable insights into the system's performance characteristics.

V. RESULTS

Our experiment aims to visualize the impact of several optimizations on the hardware implementation of RF. First, the hardware and software accuracy will be compared while keeping a full data bandwidth (in float). The latency will also be compared between these implementations. Secondly, the unroll factor will be used, and the experiments will compare implementation with and without unrolling trees while keeping the same data width. Several metrics will be displayed, such as resource usage on the FPGA (LUT, FF, DSP, BRAM), energy consumption (both static and dynamic), and latency. Thirdly, the data precision will be varied to see how it impacts resource usage and accuracy.

These metrics have been collected for RF with HP varying from 1 to 200 trees and for a maximum depth of 3 to 7. The data width has been set to float for the first experiments to objectively compare the differences between hardware and software.

Interestingly, while increasing the depth and the number of trees, the conversion time takes significantly longer. Compared to neural network implementations using HLS4ML, Conifer implementations take longer.

Table I presents the accuracy improvements achieved by fine-tuning the primary hyperparameters of the forest, specifically focusing on the number of trees (n_estimators) and the maximum depth (max_depth) of the individual trees.

In this instance, the software accuracy and hardware accuracy are nearly identical, as both involve the same precision implementation (float). The slight discrepancy can be attributed to rounding errors inherent in the implementation.

While increasing the number of estimators tends to enhance accuracy, the maximum depth plays a more decisive role. For instance, comparable accuracy can be attained using just two trees at a depth of 5, in contrast to employing 50, 100, or 200 trees with a maximum depth of 4. Conversely, latency is more significantly affected by the number of trees than by the maximum depth, both in software and hardware implementations.

The unroll RF implementation is compared to the rolled RF in Table II. This table details the percentage of FPGA resource utilization, including LUT, FF, DSP, and BRAM, as well as software and hardware latency μs, and both dynamic and static energy consumption in watts. The unrolled implementation demonstrates a significant improvement in latency, applicable to both small and large forests. Notably, in small forests (with fewer than five trees), the resource usage is lower when employing the unrolled factor. However, this trend reverses after ten trees, resulting in a doubling of resource usage with the unrolled implementation.The pattern of energy consumption exhibits a similar trend; however, it is arguably less pronounced in rolled implementations with fewer than 20 trees. Conversely, in larger forests, the significance of energy consumption becomes more pronounced in the context of unrolled implementations.

Various data precision implementations were compared on the RF implementations, as shown in Table III. The ap_fixed (a.f in the table) data type specifies a fixed-point number, where the first value denotes the total width (number of bits), including the sign bit, and the second value indicates the integer width (number of bits assigned to the integer part). Float precision is compared with three other precisions, 16,6, 12,4 and 8,2. In this instance, the precision and the unroll parameter are adjusted while different depth are studied.

The accuracy reduction in hardware is significant when using fixed-point representations of 12,4 and 8,2. While energy consumption varies, it remains relatively consistent. In contrast, latency can be reduced by half with minor precision adjustments; however, the decrease in resource usage is not as pronounced.The LUT resource is most significantly affected by changes in data precision. For instance, when using 10 trees with a maximum depth of 3, LUT utilization decreases from 7.35% in the 16.6 precision to 4.88% in the 8.2 precision, with an even lower usage of just 2.72% in floating-point implementation. Notably, despite maintaining the same examples, the latency is reduced with fixed-point implementations. With deeper forests, the float implementations consume the most resources, while decreasing the data precision also decreases the resource usage and latency.

n_estimators	max_depth	unroll	lat_skl (μs)	lat_hls (μs)	lut_prct (%)	ff_prct (%)	dsp_prct (%)	bram_prct (%)	energy_dyn (Watt)	energy_sta (Watt)	energy (Watt)
2	2	False	526.91	0.72	1.61	1.12	0.2	0.11	2.78	0.72	3.5
2	2	True	495.91	0.48	1.39	0.91	0.2	0.11	2.77	0.72	3.5
2	2	False	711.92	0.72	1.61	1.12	0.2	0.11	2.78	0.72	3.5
2	2	True	675.68	0.48	1.39	0.91	0.2	0.11	2.77	0.72	3.5
2	4	False	637.05	0.8	1.76	1.13	0.2	0.11	2.78	0.72	3.51
2	4	True	535.73	0.51	1.58	0.93	0.2	0.11	2.77	0.72	3.49
2	5	False	524.04	0.88	2.49	1.38	0.2	0.11	2.8	0.72	3.52
2	5	True	534.3	0.54	1.62	0.95	0.2	0.11	2.77	0.72	3.5
10	3	False	597.24	1.6	2.65	1.57	0.08	0.11	2.8	0.72	3.53
10	3	True	619.65	1.41	2.72	1.71	0.16	0.11	2.8	0.72	3.52
10	4	False	602.48	1.9	3.1	1.68	0.08	0.11	2.86	0.72	3.58
10	4	True	626.09	1.54	3.09	1.82	0.16	0.11	2.81	0.72	3.53
10	5	False	747.44	3.7	9.43	3.95	0.16	0.11	2.93	0.72	3.65
10	5	True	659.23	1.94	3.59	1.95	0.08	0.11	2.81	0.72	3.53
10	5	False	622.03	3.7	9.43	3.95	0.16	0.11	2.93	0.72	3.65
10	5	True	660.66	1.94	3.59	1.95	0.08	0.11	2.81	0.72	3.53
10	6	False	628.23	5.67	26.83	12.13	0.16	0.11	3.05	0.72	3.77
10	6	True	597.72	2.19	4.91	2.24	0.08	0.11	2.82	0.72	3.54
20	2	False	793.93	2.44	3.28	1.96	0.08	0.11	2.81	0.72	3.53
20	2	True	791.31	0.48	5.13	4.01	1.43	0.11	2.84	0.72	3.56
20	4	False	786.54	3.36	5.65	2.33	0.16	0.11	2.91	0.72	3.64
20	4	True	812.53	0.51	4.45	2.59	0.16	0.11	2.84	0.72	3.56
20	5	False	774.38	6.9	10.36	4.29	0.16	0.11	3.04	0.72	3.76
20	5	True	818.73	0.54	8.6	3.62	0.95	0.11	2.96	0.72	3.68
50	3	False	1251.46	4.6	5.88	2.28	0.16	0.11	2.8	0.72	3.52
50	3	True	1270.06	0.5	9.92	4.75	0.95	0.11	2.97	0.72	3.7
50	4	False	1247.17	7.56	6.69	2.53	0.16	0.11	2.98	0.72	3.7
50	4	True	1254.32	0.52	11.53	5.01	0.95	0.11	3.05	0.72	3.78
50	5	False	1243.35	16.99	13.31	4.82	0.16	0.11	3.06	0.72	3.78
50	5	True	1238.35	0.53	15.1	5.62	0.95	0.11	3.18	0.72	3.9
100	3	False	2017.97	8.6	6.69	2.46	0.16	0.11	2.98	0.72	3.7
100	3	True	1976.97	10.31	14.89	6.96	0.95	0.11	3.17	0.72	3.89
100	4	False	1968.15	16.57	8.7	2.98	0.16	0.11	3	0.72	3.73
100	4	True	1981.26	12.46	17.88	7.64	0.95	0.11	3.07	0.72	3.8
100	5	False	2001.05	33.49	18.13	5.37	0.16	0.11	3.08	0.72	3.8
100	5	True	1995.8	15.85	24.51	8.87	0.95	0.11	3.36	0.72	4.09
200	3	False	3408.43	16.59	8.71	2.82	0.16	0.11	2.99	0.72	3.71
200	3	True	3414.63	0.5	23.95	11.55	0.95	0.11	3.39	0.73	4.11
200	4	False	3428.22	32.57	13.03	3.59	0.16	0.11	3.03	0.72	3.75
200	4	True	3471.85	0.51	30.48	12.92	0.95	0.11	3.2	0.72	3.92

TABLE II
RESOURCE COMPARISON OF ROLLED/UNROLLED RF IMPLEMENTATION WITH FLOAT PRECISION

We have encountered limitations with the implementation of our RFs. When implementing an RF with a considerable amount of trees and/or a significant depth, the implementation phase can crash or take several hours to complete. This issue likely arises from the size of the tables representing the trees. It would be beneficial to divide them and see if that resolves the problem and if the latency could be improved with this approach.

VI. CONCLUSION AND PERSPECTIVES

Rather than reinventing the wheel, adapting existing high-quality work to our needs is often more accessible and more efficient. Hence, we have enhanced the Conifer framework to offer multi-class classification with Random Forest on FPGAs. Our approach presents several modifications on the conversion phase and on the writer. Nevertheless, the actual proposed extension is limited to depth of 8 for heavy datasets such as CIC-IDS2017 because of its high dimensionality and its size.

The modified framework is accessible on https://gitlabsticc.univ-brest.fr/cyber-fpga-ia/conifer_test

The prospects for implementing Random Forest on FPGA are promising, especially with the possibility of carrying out performance measurements on various datasets to compare the results obtained. This approach would make it possible to test the efficiency and accuracy of random forests on FPGAs against conventional software implementations. Comparing results obtained with different parameter configurations and datasets would provide valuable information for optimizing algorithms and improving hardware architecture. Another interesting prospect would be to improve the way decision trees are stored in arrays, which could reduce processing latency and increase resource management capacity when dealing with deep trees. These prospects pave the way for real-time applications and embedded systems where processing speed and efficiency are crucial.

n_estimators	max_depth	unroll	acc_skl	acc_hls	lat_skl	lat_hls	lut_prct	ff_prct	dsp_prct	prec
10	3	False	0.94	0.93	597.24	1.6	2.65	1.57	0.08	float
10	3	True	0.94	0.93	619.65	1.41	2.72	1.71	0.16	float
10	3	False	0.94	0.8	644.92	1.16	7.45	2.02	0.48	a.f.16,6
10	3	True	0.94	0.8	670.19	0.79	7.35	1.82	0.48	a.f.16,6
10	3	False	0.94	0.76	811.1	0.81	6.28	1.79	0.48	a.f.12,4
10	3	True	0.94	0.76	863.08	0.79	6.04	1.73	0.48	a.f.12,4
10	3	False	0.94	0.35	877.38	0.81	5.34	1.67	0.48	a.f.8,2
10	3	True	0.94	0.35	657.8	0.5	4.88	1.43	0.08	a.f.8,2
10	4	False	0.96	0.96	602.48	1.9	3.1	1.68	0.08	float
10	4	True	0.96	0.96	626.09	1.54	3.09	1.82	0.16	float
10	4	False	0.96	0.77	672.34	1.44	7.63	2.12	0.48	a.f.16,6
10	4	True	0.96	0.77	664	0.94	7.82	1.9	0.48	a.f.16,6
10	4	False	0.96	0.67	649.45	1.11	6.55	1.88	0.48	a.f.12,4
10	4	True	0.96	0.67	651.12	0.94	6.36	1.81	0.48	a.f.12,4
10	4	False	0.96	0.27	646.83	1.11	5.55	1.75	0.48	a.f.8,2
10	4	True	0.96	0.27	681.16	0.52	4.99	1.47	0.08	a.f.8,2
10	5	False	0.97	0.96	747.44	3.7	9.43	3.95	0.16	float
10	5	True	0.97	0.96	659.23	1.94	3.59	1.95	0.08	float
10	5	False	0.97	0.84	854.02	3.51	10.85	3.32	0.48	a.f.16,6
10	5	True	0.97	0.84	652.79	1.16	8.6	2	0.48	a.f.16,6
10	5	False	0.97	0.67	841.38	3.22	10.1	3.15	0.48	a.f.12,4
10	5	True	0.97	0.67	653.03	1.15	6.85	1.9	0.48	a.f.12,4
10	5	False	0.97	0.31	659.47	3.22	8.24	2.6	0.48	a.f.8,2
10	5	True	0.97	0.31	653.27	0.54	5.2	1.54	0.08	a.f.8,2
10	6	False	0.98	0.98	628.23	5.67	26.83	12.13	0.16	float
10	6	True	0.98	0.98	597.72	2.19	4.91	2.24	0.08	float
10	6	False	0.98	0.81	658.51	5.48	23.52	7.45	0.48	a.f.16,6
10	6	True	0.98	0.81	834.7	1.43	9.24	2.15	0.48	a.f.16,6
10	6	False	0.98	0.7	620.6	5.2	23.69	8	0.48	a.f.12,4
10	6	True	0.98	0.7	662.33	1.4	7.08	2.04	0.48	a.f.12,4
10	6	False	0.98	0.21	659.94	5.2	18.52	5.64	0.48	a.f.8,2
10	6	True	0.98	0.21	653.98	0.56	5.36	1.66	0.08	a.f.8,2

TABLE III
RESOURCE COMPARISON OF ROLLED AND UNROLLED RF IMPLEMENTATION WITH VARYING PRECISION FROM FLOAT TO FIXED POINT 8,2

REFERENCES

[1] Leo Breiman. "Random forests". In: *Machine learning* 45 (2001), pp. 5–32.

[2] FastML Team. *fastmachinelearning/hls4ml*. Version v0.8.1. 2023. DOI: 10.5281/zenodo.1201549. URL: https://github.com/fastmachinelearning/hls4ml.

[3] *Vitis Unified Software Platform*. https://www.xilinx.com / products / design - tools / vitis . html. Accessed: [current date].

[4] Sioni Summers et al. "Fast inference of boosted decision trees in FPGAs for particle physics". In: *Journal of Instrumentation* 15.05 (2020), P05026.

[5] Gérard Biau and Erwan Scornet. "A random forest guided tour". In: *Test* 25 (2016), pp. 197–227.

[6] Yisen Wang et al. "A novel consistent random forest framework: Bernoulli random forests". In: *IEEE transactions on neural networks and learning systems* 29.8 (2017), pp. 3510–3523.

[7] Lior Rokach. "Decision forest: Twenty years of research". In: *Information Fusion* 27 (2016), pp. 111–125.

[8] Fabian Pedregosa et al. "scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011). Accessed on September 16, 2024, pp. 2825–2830. URL: https://scikit-learn.org/stable/.

[9] Flora Amato et al. "An fpga-based smart classifier for decision support systems". In: *Intelligent Distributed Computing VII: Proceedings of the 7th International Symposium on Intelligent Distributed Computing-IDC 2013, Prague, Czech Republic, September 2013*. Springer. 2014, pp. 289–299.

[10] Muhsen Owaida et al. "Scalable inference of decision tree ensembles: Flexible design for CPU-FPGA platforms". In: *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE. 2017, pp. 1–8.

[11] Hiroki Nakahara et al. "A random forest using a multi-valued decision diagram on an FPGA". In: *2017 IEEE 47th international symposium on multiple-valued logic (ISMVL)*. IEEE. 2017, pp. 266–271.

[12] Rastislav Struharik. "Decision tree ensemble hardware accelerators for embedded applications". In: *2015 IEEE 13th International Symposium on Intelligent Systems and Informatics (SISY)*. IEEE. 2015, pp. 101–106.

[13] Junjie Bai, Fang Lu, Ke Zhang, et al. *ONNX: Open Neural Network Exchange*. https://github.com/onnx/onnx. 2019.

[14] Andreas Hoecker et al. "TMVA-toolkit for multivariate data analysis". In: *arXiv preprint physics/0703039* (2007).

[15] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[16] Mathieu Guillame-Bert et al. “Yggdrasil Decision Forests: A Fast and Extensible Decision Forests Library”. In: *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*. 2023, pp. 4068–4077. DOI: 10.1145/3580305.3599933. URL: https://doi.org/10.1145/3580305.3599933.

[17] *PYNQ — Python Productivity to AMD Adaptive Compute Platforms*. http://www.pynq.io/. Accessed on February 20, 2024.